

# 07 OOP - その1

作成者 : GT F

最終更新日 : 2019-08-17

- [オブジェクト指向プログラミング \(OOP\)](#)
- [型 \(Class型\)](#)
- [クラスとオブジェクト](#)
  - [クラス定義の方法](#)
  - [クラス定義注意](#)
  - [クラスメンバ](#)
- [クラスの使用](#)
  - [インスタンスの生成](#)
  - [クラスの使用](#)
- [メンバアクセス制御](#)
- [UML](#)
- [NullPointerException](#)
- [this キーワード](#)
- [Getter & Setter](#)
- [既存クラスの利用](#)
- [質問](#)
- [初心者よく間違いところ](#)
- [コンストラクタ](#)
  - [暗然的なコンストラクタ](#)
  - [明示的なコンストラクタ](#)
  - [コンストラクタの補足](#)
  - [final ファイナル](#)
  - [final ファイナル + コンストラクタ](#)
  - [複数のコンストラクタ](#)
  - [静的のメンバ](#)
- [よく使用する静的メソッド](#)
- [質問 :](#)
- [質問のヒント](#)
- [概念理解](#)
- [覚えておこう](#)

## オブジェクト指向プログラミング (OOP)

オブジェクト指向では、現実世界のあらゆる「もの」を、それが物理的に存在するもの（たとえ商品、顧客）であっても、概念的なもの（たとえば注文、契約）であっても、すべてオブジェクトとして認識しています。オブジェクトとは、そのオブジェクトの性質を表す属性という部分と、オブジェクトの動作や振り舞いを表す操作という部分で構成されると考えます。

## 型（Class型）

クラス型とインスタンス化

復習

- ・ 数値型
- ・ 文字列型
- ・ 配列型

人間と考えます。人間はの型は？ 人間の名前は型は？ 年齢は？

- ・ クラス型（カスタマイズ customize 型）複数型。

## クラスとオブジェクト

クラス ⇒ 人間の概念（**抽象的**）。アジア人、アメリカ人...⇒ 抽象的を**具象化**したもの。

## クラス定義の方法

メソッドはデータを持っています。データに対する処理をメソッドで記述する。データをデータに対する処理はクラスメンバと呼ばれます。

```
1 package earth: //所属のパッケージを定義する
2 修飾子 class クラス名 {
3 }
```

例：人間型を定義する。

```
1 package earth: // 同じパッケージに、クラス名重複不可
2 public class Human {
3 }
```

新しいの複合型 `Human` を定義します。該当クラスに2の属性が含まれています。文字列の名称、整数型の年齢であること。コデイン規則通り、クラスの名頭文字は大文字の半角数字。例 `String`, `Math`。クラスを利用するため、インスタンス化必要があります。

## クラス定義注意


1. 原則に**1つクラス**と**1つのファイル**に定義する。
2. **クラス名 = ファイル名**（異なる場合コンパイルエラーが発生します）
3. クラス名先頭文字は大文字の**半角英字**。
4. 同じパッケージにクラス名**重複不可**

 クラス名変更 & Package変更する場合（IDE：Quick Fix）

クラス名変更する場合ファイル変更する必要があります。又はPackage名変更する場合、クラス移動する必要があります。いちいちファイル操作面倒くさいので、IDEはQuick Fixという自動的にリファクタリング機能が付けています。

例1：クラス変更するQuick Fix

例2：パッケージ移動のQuick Fix

 内部クラス（後編）以外、1クラスは1 Javaファイルであること。Javaファイル名はクラス名と同じ。（大文字小文字意識する必要があります）

## クラスメンバ

クラスメンバは2種類持っています。1. **属性（フィールド Field）**、2. 属性に対する**操作（関数・メソッド）**

```
1 public class Human {
2     // 属性：名称を定義する
3     public String name;
4     // コンストラクタ
5     public Human() {
6         // コンストラクタに初期化処理を行う
7         // コンストラクタ詳細は後で説明します。
8         name = "名前なし";
9     }
10    // 属性名称に対する操作
11    public String getName() {
12        return name; // 該当クラスの属性：名称を返します。
13    }
14 }
```

Humanというクラス2つメンバもっています。

1. String型のフィールドname, 初期値は "名前なし"
2. フィールドnameを取得するメソッド（操作）getNameをもっています。

## クラスの使用

### インスタンスの生成

インスタンス(instance)。抽象的クラスを具象化する。例は人間の概念を具体的にアジア人とはアメリカ人とか具象化する。

変数宣言：

```
1  int a; // 基本数値型変数aを宣言する
2  Human usa; // 複数型bを宣言する
3  Human asia;
```

値の代入

```
1  a = 10;
2  usa = new Human(); // キーワード new + コンストラクタ();
3  asia = new Human();
```

クラス型を利用するために、インスタンス化（instance）の必要があります。インスタンス生成するには new 演算子を使用します。

## クラスの使用

演算子 . で、クラスの属性へアクセスします。基本構文は以下

- クラスタイプの変数名 . 属性
- クラスタイプの変数名 . メソッド(引数...)

クラスの属性とメソッドアクセスの一例：

```
1  System.out.println(b.name); // bの名前をprintする。⇒名前なし
2  System.out.println(b.getName()); // クラスのメソッドを呼び出す。⇒名前なし
3  Dog dog = new Dog(); // Dog型をインスタンス化する。
4  String name = dog.name; // Dogの名前を取得する。
5  dog.eatShit(); // Dogを**食べさせる。
```

= で、クラスの属性に値を代入する

```
1  b.name = "DCNET"; // bの名前に文字列"DCNET"を代入する
2  System.out.println(b.name); // DCNET
```

※プログラミングにアクセスする意味は「値をを取得するまた値を設定する」「メソッドを呼び出す」であること。

# メンバアクセス制御

クラスにてすべての属性・メソッドはアクセス可能ではありません。

アクセス修飾子

1. public : 公開 (+)
2. protected : 保護 (#)
3. private : 私有 (-)

アクセス修飾子が省略された場合、privateとなります。

質問：以下コードは**コンパイルエラー**発生しますか？理由は？

```
1 // クラス 1 Human.java
2 package earth;
3 public class Human {
4     private String name;
5     protected int age = 10;
6     public String getName() {
7         return name;
8     }
9 }
10 // クラス 2 Sample.java (ファイル異なる要注意)
11 package earth;
12 public class Sample {
13     public static void main(String...args) {
14         Human man = new Human();
15         System.out.println(man.name); // NG, nameはprivateの為、アクセス不可
16         System.out.println(man.age); // OK, ageはprotected、同じPackageのクラスアクセス可能
17         System.out.println(man.getName()); // OK, publicの為、いつでもどこでもアクセス可能
18     }
19 }
```

## UML

UML(Unified Modeling Language : 統一モデリング言語)は、ソフトウェア設計とパターンを表現するための視覚的な言語です。UMLはいくつ種類があります。

代表的なUML図

1. クラス図
2. パッケージ図
3. コンポジット構成図
4. コンボネット図
5. 配置図
6. ユースケース図
7. 状態マシン図

## 8. アクティブディ図

## 9. 相互作用図（そう ごさよう）

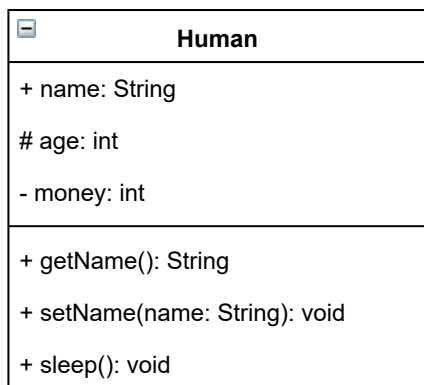
## 10. ER図

※強調しているアイテムは必須知識であること。

クラス図（class diagram）は、UMLの最も基本的な図の1つです。クラス図は、共通の状態と振る舞いを持つものの集まりを表します。クラスは長方形として表現され、その中は3つの区画に分けられます。

1. 第1区画：クラス名称を記述しています。
2. 第2区画：クラスの属性（フィールド）を記述しています。
3. 第3区画：クラスのメソッド（関数）を記述しています。

下記クラス記述の一例です。



質問：上記UML図通り、Javaクラスを作成してください。「**private**」、「**protected**」、「**public**」等キーワードを覚えておいてください。

# NullPointerException

参照型（クラス型・オブジェクト型）の初期値を `null` に設定してください。インスタンス化しないままオブジェクトをアクセスまたはメソッドを呼び出すとき、NullPointerException（ヌルポインター）の例外が発生します。NullpointerはJavaプログラミングによく発生している例外である。

質問：下記コードのBugを探します。

```
1 // Human.java
2 public class Human {
3     private String name;
4 }
5 // Kicker.java
6 public class Kicker {
7     public static void main(String... args) {
```

```
8         Human obj = null;
9         System.out.println(obj.name); // Nullpointer : obj = null;
10     }
11 }
```

## this キーワード

`this` は、自分の**インスタンス**を指すキーワードです。インナークラス内で`this`を使用した場合、インナークラスのインスタンス自身や、インナークラスのインスタンス内のメンバを指します。メソッド内にメンバ変数と同じ名前のローカル変数がある場合、ローカル変数が優先し利用されます。このとき、「`this.変数名`」と指定すると、インスタンスないのメンバ変数を指定することができます。

```
1 public class People {
2     private String name;
3     public String getName() {
4         // ここに this 省略可能
5         return name;
6     }
7     // 下記メソッドは引数のnameの値をクラスメンバ #2 nameに代入する
8     public void setName(String name) {
9         name = name; // 引数の name = 引数の name。意味ないステートメント
10        this.name = name; // this の意味現在のクラスのオブジェクト
11    }
12 }
```



変数名をクリック、highlight所は同じ変数を示します。（IDE機能）

## Getter & Setter

Javaでは、直接にクラスの属性するを…。getter + setter メソッドを作成して属性をアクセスする

```
1  public class Human {
2      /** 名称：私有を設定して */
3      private String name;
4      /**
5       * 名称を取得する
6       * @return 名称
7       */
8      public String getName() {
9          return name;
10     }
11     /**
12     * 名称を設定する
13     * @param name 設定した名称値
14     */
15     public void setName(String name) {
16         this.name = name;
17     }
18 }
19 // ....
20 public static void main(String...ars) {
21     Human obj = new Human();
22     // publicの場合
23     obj.name = "xxx";
24     // Getter & Setter の場合
25     String name = obj.getName(); // String name = obj.name
26     obj.setName("new Name"); // obj.name = "xxx";
27 }
```

IDE基本より、Getter & Setter を自動作成する可能です。

クラス編集に、**右クリック** > **コンテンツメニュー** > **ソースを挿入** > **Getter及びSetterメソッドをジェネレータ** > 作成しようフィールドを選択して「OK」ボタンをクリックする。

## 既存クラスの利用

他人定義済みクラスを利用したい場合、クラス参照と呼びます。同一パッケージ又は `java.lang` パッケージのクラス群を直接に参照可能です。この以外の場合、`import` キーワードを使用して、クラスをインポートする必要があります。

使い方法は以下：

```
1  import クラスのパッケージ.クラス名
```

使う例：

```
1  package firstproject;
2  // ↓ java.util.Date をインポートする
```



```
3 import java.util.Date;
4 public class PkgImp {
5     public static void main(String[] args) {
6         Date date = new Date();
7         System.out.println(date);
8     }
9 }
```

Netbeans又Eclipseで、`ctrl` + `shift` + `o`でパッケージを一括インポートできます。

## 質問

質問 1 : ペットクラスを定義してください。ペットクラスは名称と年齢を保有して。以下 2 インスタンスを作成してください。変数 1 : cat , 変数 2 : dog。catの名称は "cat", 年齢 =2, dogの名称は "dog", 年齢=5。

1. クラスを定義する。クラス名は Pet。
2. 該当クラスに、「名称」「年齢」属性が保有しています。
3. Petクラスをインスタンス化する。以下 2 つオブジェクトを作成してください。
4. オブジェクト 1 : 変数名 = cat。名称 = "cat", 年齢 = 2。
5. オブジェクト 2 : 変数名 = dog。名称 = "dog", 年齢 = 5。

 クラスとオブジェクトの区別を理解してください。

クラスとは抽象的に定義であること。オブジェクトはクラスを現実世界の反映である。同じクラス型のオブジェクトにて、差異があります。上記の cat, dogはPetクラスのインスタンスであり。属性がそれぞれ違っています。`cat.age = 2`, `dog.age = 5`。

質問 2 : アマゾンの各商品(Product)にて、クラスを定義してください。例 : 価格、写真、商品名など...

## 初心者よく間違いところ

間違いやすい所 1 : メソッド呼び出すとフィールドアクセス誤りました。#14行, #15行の違い? Petクラスにどのフィールド値が変更されましたか?

```
1 // Pet.java
2 public class Pet {
3     public String setName;
4     public String name;
5
6     public void setName(String value) {
7         this.name = value;
8     }
```

```

9  }
10 // Kicker.java
11 public class Kicker {
12     public static void main(String...args) {
13         Pet cat = new Pet();
14         cat.setName = "xxx"; // フィールド?メソッド?
15         cat.setName("xxx"); // フィールド?メソッド?
16     }
17 }

```

間違いやすい所2：定義しないメソッド／フィールドをアクセスする

```

1  // Pet.java
2  public class Pet {
3      public String name;
4
5      public void setName(String value) {
6          this.name = value;
7      }
8  }
9  // Kicker.java
10 public class Kicker {
11     public static void main(String...args) {
12         Pet cat = new Pet();
13         cat.setAge(1234); // 定義されないメソッドsetAge(int)をアクセスする。
14     }
15 }

```

## コンストラクタ

通常、クラスをインスタンス化した直後には、インスタンスのメンバ変数に値を設定しています。コンストラクタを利用すると、インスタンス生成時に値の初期化などの処理をおこなうことができます。すべてのクラスは暗然的なコンストラクタは保有しています。暗然コンストラクタが省略可能です。コンストラクタは特殊のメソッドです。

 コンストラクタはクラスメンバの値を初期化をおこなう。

コンストラクタの宣言は以下ルールがあります。

1. コンストラクタは**メソッド（関数）**であること。
2. コンストラクタの**メソッド名はクラス名**と同じ。（大文字と小文字を区別する）
3. コンストラクタは**戻り値を記載不要**です。（void）も必要ありません。

コンストラクタの例

```

1  public class People {
2      public String name = "undefined";
3      /** コンストラクタ*/

```

```
4     public People() {
5         this(10); // コンストラクタからコンストラクタを呼び出す
6     }
7     /** コンストラクタ*/
8     private People(int a) {
9         System.out.println("引数があるコンストラクタ");
10        this.doInit(); // 別メソッドを呼び出す
11    }
12    // 普通のメソッド
13    public void doInit() {
14        name = "初期化します...";
15    }
16 }
```

## 暗然的なコンストラクタ

すべてのクラスは引数0のコンストラクタが保有しています。インスタンスする構文 `People obj = new People();`。⇒ キーワード `new` の直後はコンストラクタを呼び出します。

以下クラスは暗然（隠された）なコンストラクタを保有しています。

1. 式の左辺：変数宣言 `People obj;`
2. 式の右辺：コンストラクタを呼び出し、インスタンス化する。**`new People();`**
3. 演算子`=`：右辺の式の値を左辺に代入する。

## 明示的なコンストラクタ

以下のクラスが定義されているとします。明示的なコンストラクタを定義する場合、暗然のコンストラクタが自動的に無効される。なお、クラスのコンストラクタには引数を渡すことができます。

```
1 // People1.java
2 public class People1 {
3     /**属性*/
4     public String name;
5     /**明示的なコンストラクタ：引数1*/
6     public People1(String name) {
7         this.name = name;
8     }
9 }
10 // Kicker.java
11 public class Kicker {
12     public static void main(String...args) {
13         People1 obj = new People1(); // コンパイルエラー
14     }
15 }
```

この場合：`People1 obj = new People1();` はビルドエラー発生します。理由はコンストラクタは1つ引数が必要です。コンストラクタはオブジェクトのライフサイクルに1回のみ実施する。

```
1 public class People2 {
2     public String name;
3 }
```

明示的なコンストラクタがある場合：

```
1 People1 p1 = new People1("DCNET");
```

明示的なコンストラクタがない場合：

```
1 People2 p2 = new People2();
```

`this` キーワードを用いコンストラクタは他のコンストラクタ呼び出すのは可能です。

```
1 public class Sample {
2     public Sample(String m) {
3         System.out.println("クラスを初期化する。");
4     }
5     public Sample(int m) {
6         // 他のコンストラクタを呼び出す場合、必ず1行目
7         this("他のコンストラクタを呼び出す");
8         int a = 10;
9     }
10 }
```

## コンストラクタの補足

1. コンストラクタは演算子 `new` の直後実行されること。
2. コンストラクタの名前は必ずクラス名に同じにする。
3. 明示的なコンストラクタを作成しない場合、黙然な引数なしのコンストラクタを自動的に作成されること。
4. 1つクラスに複数のコンストラクタ作成可能。

## final ファイナル

基本は宣言された変数に、何回値代入する問題がありません。

```
1 int a = 0;
2 a = 10;
3 a = 20;
```

但し、`final` が修飾された変数は1回のみ値変更可能です。ファイルの変数は**必ず**値を設定する必要があります。

```
1 final int a = 0;
```

```
2  a = 100; //コンパイルエラー
3  final String x = null;
4  x = "new string"; //nullでも、変更不可。
```

質問：以下コンパイルエラー理由を教えてください。

```
1  public class Sample {
2      private final int a = 10;
3      public void setA(int a) {
4          this.a = a; // NG フィールドaの値を再代入しよう
5      }
6  }
```

## final ファイナル + コンストラクタ

final フィールドの値を代入する箇所は**コンストラクタ**のみです。

```
1  public class Sample {
2      public final int age = 10;
3      public final String name;
4      public Sample(String name) {
5          this.name = name; // nameは必ずコンストラクタに値を代入する必要があります
6      }
7  }
```

## 複数のコンストラクタ

複数のコンストラクタは定義可能です。

```
1  public class People {
2      public String name;
3      /** 引数ないコンストラクタ*/
4      public People() {}
5      /** 引数ありコンストラクタ */
6      public People(String name) {
7          this.name = name;
8      }
9  }
```

## 静的のメンバ

静的のメンバは**インスタンスしなくても**アクセスする可能です。アクセス方法は クラス名.**.**属性。静的メンバは一般的に共通Utilsメソッド、定数など実装します。静的メンバの取扱十分注意する必要があります。（Java高級：スレッドセーフに説明します。）静的のメンバはアプリケーション全体変数、ライフサイクルは最長、アプリの起動から終了までメモリ上に存在する。非静的のメンバはオブジェクトを**new**（インスタンス化）時代誕生、オブジェクトを破棄する場合アクセス不能。

静的メンバの宣言は修飾子 `static` を追加します。

```

1 // Human.java
2 public class Human {
3     public static final String PLANT = "地球";
4     public static void staticMethod() {
5         System.out.println("静的メソッド");
6     }
7     public void sample() {
8         System.out.println("非静的メソッド");
9     }
10 }
11 // Kicker.java
12 public class Kicker {
13     public static void main(String...args) {
14         System.out.println(Human.PLANT); // 静的なフィールドにアクセス。
15         Human.staticMethod(); // 静的なメソッドにアクセス
16         // Human obj = new Human();
17         // obj.sample();
18         new Human().sample(); // 非静的メソッドアクセス、インスタンス化必要。
19     }
20 }

```

静的なメンバのアクセス方法は：クラス名.属性名 or クラス名.メソッド名(引数)

IDEの操作：

1. フィールド 又は メソッド呼び出すところ： `Ctrl` + 左クリック ⇒ 定義するところへ飛び出す。
2. `ALT` + `←` 前のドキュメントへ移動
3. `ALT` + `→` 後のドキュメントへ移動

質問：以下メソッドの静的メンバアクセス所を説明してください。（1箇所）

```

1 public static void main(String...args) {
2     System.out.println("some thing"); // ここに1箇所静的メンバアクセスしている
3 }

```

1. 静的なフィールドは？ 原因は？
2. メソッドを呼び出していますか？
3. メソッドを呼び出す時、渡すの引数は？

⚠ 静的のメンバは静的のメンバのみアクセス可能。

2. 以下ソースのコンパイルエラーを説明してください。

```

1 public class Kicker {
2     private String word = "abc";
3     public void print(String name) {
4         System.out.println(name);

```

```
5    }
6    public static void main(String...args) {
7        print("dcnet. java. 教育");
8        System.out.println(word);
9    }
10 }
```

3.質問 2 に以下ソース修正したらコンパイルエラー解消できますが、理由を教えてください。

```
1  public class Kicker {
2      private String word = "abc";
3      public void print(String name) {
4          System.out.println(name);
5      }
6      public static void main(String...args) {
7          Kicker kicker = new Kicker();
8          kicker.print("dcnet. java. 教育");
9          System.out.println(kicker.word);
10     }
11 }
```

# よく使用する静的メソッド

クラスおよびメソッド	例	説明
基本型操作		
Integer.valueOf(String x)	int x = Integer.valueOf("001");	文字列から数値を変換する。 parseInt 同様
Float.valueOf(String x)	float y = Float.valueOf("0.1F");	文字列から数値を変換する。
Boolean.valueOf(String x)	boolean b = Boolean.valueOf("true");	文字列から論理値を変換する。
String.valueOf(...)	String v = String.valueOf(true);	引数は何でもいいです。
Math （数学）		
Math.min(float x, float y)	float y = Math.min(1.0F, 2.0F);	MINを取得する
Math.max(float x, float y)	float y = Math.max(1.0F, 2.0F)	MAXを取得する
Math.abs(int a)	int y = Math.abs(-1.0F)	絶対値
Math.round(float x)	float z = Math(1.49999F);	四捨五入

Math.floor(float x)	float z = Math(1.99999F);	切捨て
Math.ceil(float x)	float z = Math(1.00001F);	切り上げ

※すべての基本型は対応するオブジェクト型が存在します。

- int -> Integer
- float -> Float
- double -> Double
- boolean -> Boolean
- byte -> Byte
- char -> Char

質問：レコードが123行があります。各ページは50行を表示します。レコードをすべて表示する為、ページ数を求めてください。

## 質問：

質問1：以下コードのコンパイルエラー理由を教えてください。

```

1  // Pet.java
2  public class Pet {
3      public String name;
4
5      public void setName(String value) {
6          this.name = value;
7      }
8  }
9  // Kicker.java
10 public class Kicker {
11     public static void main(String... args) {
12         Pet cat = new Pet();
13         cat.setAge(1234);
14     }
15 }
```

質問2：Mathクラスを作成してください。以下メソッドを定義します。

1. 配列のMAXを求める**静的**メソッドを定義。（引数は int 配列）
2. 配列のMINを求める**静的**メソッドを定義。（引数は int 配列）
3. 配列の平均値を求める**静的**メソッドを定義。（引数は int 配列）
4. mainメソッドで上記それぞれメソッドを呼び出してください。

以下はメソッドの定義のソースの抜粋です。

```

1  public class Math {
2      public static int max(int[] input) {
3          //...
```



```

4     }
5 }

```

2.行列計算。2D行列クラスのコンストラクタには1 配列引数設定してください。

```

1  public class Matrix {
2      private final int[][] data;
3      /** construct method */
4      public Matrix(int[][] data) {
5          this.data = data;
6      }
7      // 行列 ADD
8      public Matrix add(Matrix b) {
9          //....
10         return ???
11     }
12 }

```

ヒント：行列の加算は以下通り



## 質問のヒント

### 行列計算の考え方法 1

1. 新規型 `Matrix` を定義します。
2. 行列の加算を考えます。Matrix a, Matrix b, Matrix c = a + b;
3. 但し算術演算子は基本型（数値型）のみ利用可能の為、addメソッドを定義する必要。
4. addメソッド：引数 1：タイプ行列、戻り値の型：行列（Matrix）⇒ `this` をよく理解してください。

```

1  public Matrix add(Matrix b) {
2      Matrix c = new Matrix();
3      c.data[0][0] = this.data[0][0] + b.data[0][0];
4      c.data[0][1] = this.data[0][1] + b.data[0][1];
5      c.data[1][0] = this.data[1][0] + b.data[1][0];
6      c.data[1][1] = this.data[1][1] + b.data[1][1];
7      return c;
8  }

```

### 行列計算の考え方法 2

```

1  // Matrix.java
2  public class Matrix {
3      private int[][] data
4  }

```

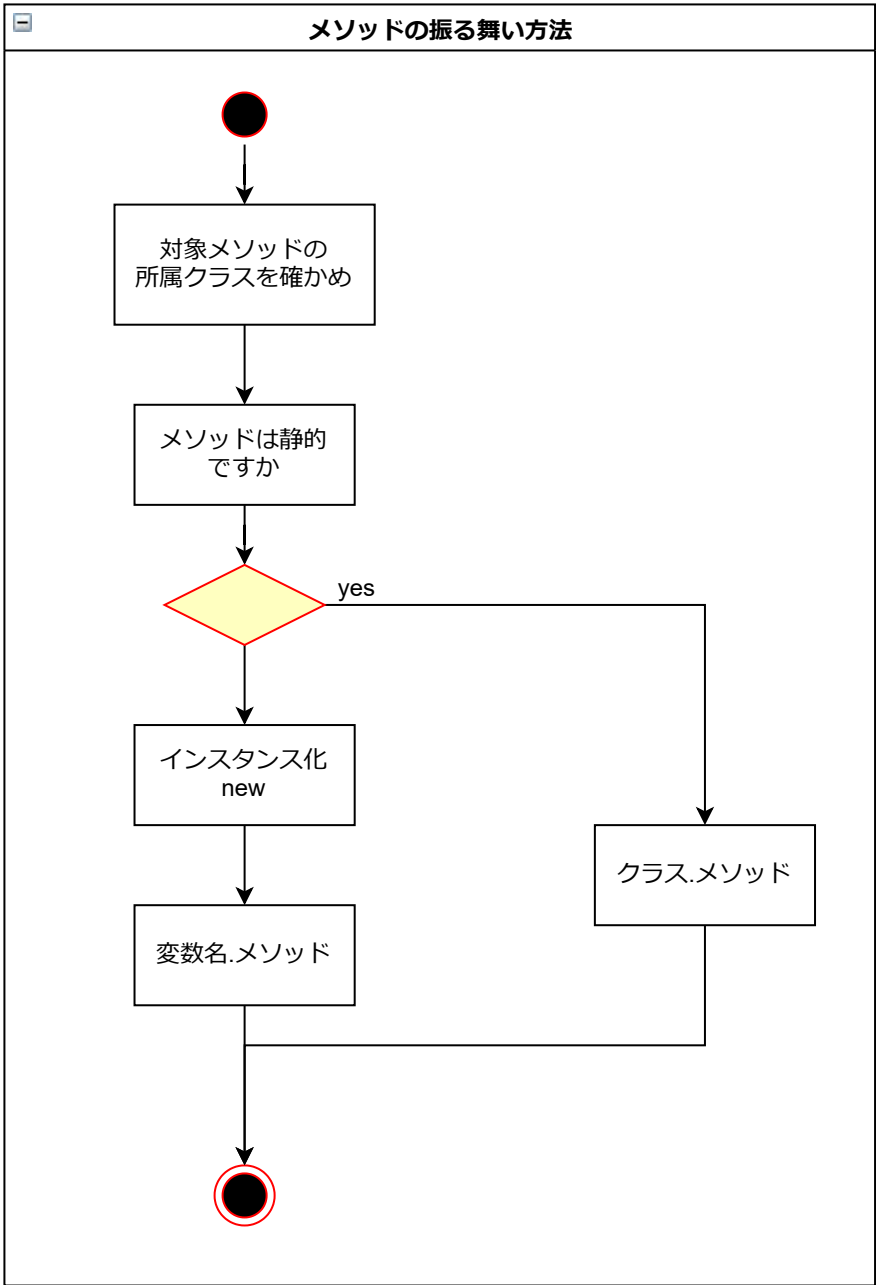
```
5 // Kicker.java
6 public class Kicker {
7     public static Matrix add(Matrix a, Matrix b) {
8         Matrix c = new Matrix();
9         c.data[0][0] = a.data[0][0] + b.data[0][0];
10        // ...
11        return c;
12    }
13 }
```

## 概念理解

1. Classは**型**です。型は**抽象的**なもの。Javaにすべてのメソッド・フィールドは所属クラスがあり。
2. Classをインスタンス化した後はオブジェクトObject。インスタンスのキーワードは `new`。オブジェクトは**具象的**なもの。
3. Classのメンバ。
4. Classのメンバのアクセス制御（3種類 +, #, -）
5. Classの属性を初期するメソッドはコンストラクタ。
6. 静的 `static` メンバアクセスのはインスタンス化不要。
7. Package
8. Classの命名規則。
9. クラスの属性(Field)を中心して、すべてのメソッドは属性に対応操作です。
10. コンストラクタは属性の初期化を行います。
11. Getterは属性を取得する操作です。
12. Setterは属性に値を設定する操作です。
13. コンストラクタを複数存在して、オブジェクトをインスタンス化する複数の選択肢を提供します。
14. 静的なメソッドを静的なフィールド・メソッドのみアクセス可能
15. 静的なメソッドを非静的なフィールド・メソッドをアクセスしたい場合、所属クラスをインスタンス化する必要があります。

## 覚えておこう

クラスメンバのアクセス方法。



いいね 1 番に「いいね」しましょう

ラベルがありません